

# 简易文本编辑器

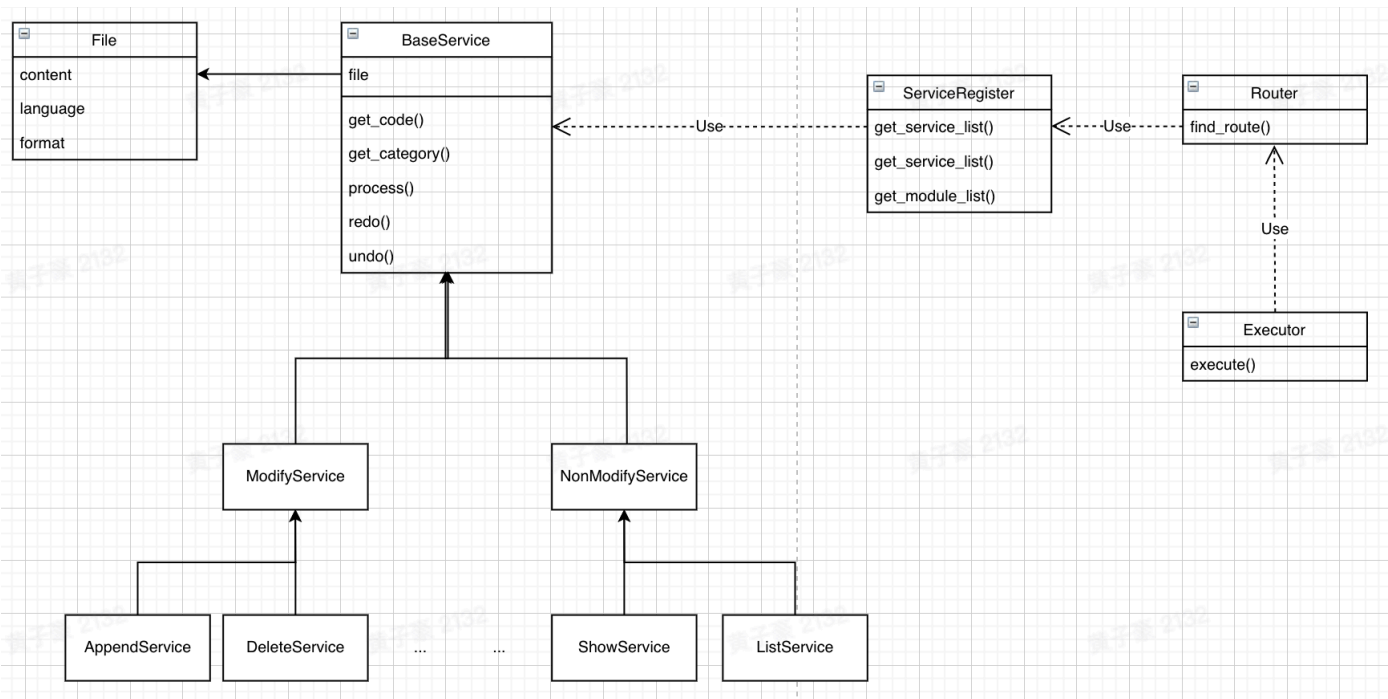
By Huang Zihao 18302010034

## 简易文本编辑器

- 程序结构
- 设计模式
  - 单例模式
  - 装饰器模式
  - 观察者模式
- 我的设计模式的优点
- Why Python Not Java?
- For TA: How To Test?

## 程序结构

程序图: <https://bytedance.feishu.cn/docx/doxcnggJcKeAh5ZtxCJZybTSVdh>



Everything is object. 在我的设计中，不仅具体的file是对象，连路由模块Router、ServiceRegister和每条指令都是一个对象。

每条指令都继承自抽象的BaseService，并implements它的五个函数。

- get\_code用于路由模块
- get\_category用于判断需不需要redo和undo
- process每条命令处理具体的工作
- undo和redo负责对于修改类命令undo和redo指令对象自己

ServiceRegister通过反射技术获取BaseService的所有子类，并提供给路由模块调用。

Route根据指令的开头判断调用BaseService的哪个实例。

## 设计模式

---

### 单例模式

修改的文件对象采用了单例模式，确保了所有service对象修改的是同一个文件。

### 装饰器模式

实现了装饰器singleton\_wrapper和exception\_wrapper，因为很多场景下都需要单例模式和异常处理，使用装饰器模式能极少地减少对程序代码的侵入性，更加模块化。

### 观察者模式

宏命令的实现刚开始着实令我费了一些脑筋，因为ServiceRegister通常都是程序初始化的时候主动的去获取所有的服务指令的，如何在程序运行中动态地添加新的指令呢？我的解决方案是：不是ServiceRegister定时地去“拉取”所有服务指令，而是把ServiceRegister设置成观察者，每当服务指令更新时去通知ServiceRegister。

## 我的设计模式的优点

---

1. 便于扩展，这是最大的优点。当新增指令时，只需要在service模块下新增一个service子类，实现它的5个接口（每一个的实现都很简单，只需要关注指令本身实现功能！）
2. 便于理解，任何看过代码的人能很快捷地理解这个项目

## Why Python Not Java?

---

python和java都是面向对象的语言，通过我的程序结构图可以很清楚的发现在我的程序中oop的思想无处不在：

- 万物皆对象。每种指令、Router、ServiceRegister和File都是对象
- 继承。所有指令继承自BaseService

而选择python的原因是因为python中使用装饰器和反射非常方便：可以通过注解的形式方便地引用一个装饰器，可以通过module模块相关api方便地实现ServiceRegister。

所以我选择python，这可以很好地实现我的程序结构，体现oop的思想，并且python有着方便的装饰器和反射。

## For TA: How To Test?

---

```
test.py x
37
38 ▶ if __name__ == "__main__":
39     # 这6个auto_test分别对应 1个自动测试用例 和 5个手工测试用例，取消和加上对应的注释语句即可
40     auto_test('自动测试用例/AutoTestCase.txt', '自动测试用例/AutoTestCaseOutput.txt', '自动测试用例/AutoTestCaseWant.txt')
41
42     # auto_test('手工测试用例/1-命令列表部分/TestCase01.txt', '手工测试用例/1-命令列表部分/TestCase01Output.txt',
43     #         '手工测试用例/1-命令列表部分/TestCase01Want.txt')
44     # auto_test('手工测试用例/1-命令列表部分/TestCase02.txt', '手工测试用例/1-命令列表部分/TestCase02Output.txt',
45     #         "手工测试用例/1-命令列表部分/TestCase02Want.txt")
46     # auto_test('手工测试用例/2-拼写检查部分/TestCase01/TestCase01.txt', "手工测试用例/2-拼写检查部分/TestCase01/TestCase01Output.txt",
47     #         "手工测试用例/2-拼写检查部分/TestCase01/TestCase01Want.txt")
48     # auto_test('手工测试用例/2-拼写检查部分/TestCase02/TestCase02.txt', "手工测试用例/2-拼写检查部分/TestCase02/TestCase02Output.txt",
49     #         "手工测试用例/2-拼写检查部分/TestCase02/TestCase02Want.txt")
50     # auto_test('手工测试用例/3-Bonus/TestCase01/TestCase01.txt', "手工测试用例/3-Bonus/TestCase01/TestCase01Output.txt",
51     #         "手工测试用例/3-Bonus/TestCase01/TestCase01Want.txt")
52
```

运行项目根目录下的test.py文件即可测试自动测试用例和5个手工测试用例，如果要测试新的测试用例，只需要替换掉测试的文本和字典即可

## 自动测试用例说明



```
1 a "What is your point "  
2 A "of view"  
3 s  
4 D 5  
5 u  
6 s  
7 r  
8 s  
9 u  
10 s  
11 d 5  
12 u  
13 s  
14 l 10  
15 s  
16 m 2 m2  
17 s  
18 $m2  
19 lang eng  
20 content txt  
21 spell  
22 spell-a  
23 spell-m  
24
```

这个测试用例覆盖了所有的指令，包括A a s D d u r l m lang content spell spell-a spell-m